ELSEVIER

Theoretical Computer Science 197 (1998) 157–169

Theoretical Computer Science

An O(n) time hierarchical tree algorithm for computing force field in n-body simulations

Guoliang Xue*

Department of Computer Science and Electrical Engineering, College of Engineering and Mathematics, The University of Vermont, Burlington, VT 05405-0156, USA

Received May 1997
Communicated by D.-Z. Du

Abstract

We consider the following force field computation problem: given a cluster of n particles in three-dimensional space, compute the force exerted on each particle by the other particles. Depending on different applications, the pairwise interaction could be either gravitational or Lennard–Jones. In both cases, the force between two particles vanishes as the distance between them approaches to infinity. Since there are $n(n-1)/2$ pairs, direct method requires $\Theta(n^2)$ time for force-evaluation, which is very expensive for astronomical simulations. In 1985 and 1986, two famous $O(n \log n)$ time hierarchical tree algorithms were published by Appel (1985) and by Barnes and Hut (1986), respectively. In this paper, we show that Appel's algorithm can be implemented in $O(n)$ time.

Keywords: Spatial tree algorithms; Force field evaluation; n-body simulations

1. Introduction and assumption

Fast algorithms for force field evaluation have important applications in molecular conformation, molecular dynamics, and astrophysical simulations. Given a cluster of n particles in three-dimensional space, we need to compute the force exerted on each particle by the other particles. Since there are $n(n-1)/2$ pairs, direct method requires $\Theta(n^2)$ time for force-evaluation, which is very expensive for astronomical simulations.

In astrophysical simulations, the force exerted by one particle on another is given by the gravitational force. In molecular dynamics and molecular conformation, the Lennard–Jones potential is widely used. In both cases, the force exerted on one particle by another particle vanishes as the distance between them approaches to infinity.

* Corresponding author. E-mail: xue@cs.uvm.edu.

0304-3975/98/$19.00
PII S0304-3975(97)00232-6

This observation leads to several fast approximation algorithms. In 1985 and 1986, two famous $O(n \log n)$ time hierarchical tree algorithms were published by Appel [2] and by Barnes and Hut [3], respectively. In 1987, Greengard and Rokhlin [8] published the fast multipole algorithm which computes the force field in $O(n)$ time. These algorithms have made great impacts on the computational study of molecular conformation/dynamics and astronomical simulations. Due to the big constant in the fast multipole algorithm and the simplicity and efficiency of the tree algorithms, hierarchical tree algorithms received more attention in computational studies [1]. Therefore, we concentrate on tree algorithms in this paper.

A central idea behind Appel's algorithm and the Barnes–Hut algorithm is the *monopole approximation* [2]. Appel showed that when a group of n_1 particles and a group of n_2 particles are *well separated* from each other, we can approximate the $n_1 \times n_2$ pair interactions by a single-pair interaction between two *big* particles (one at the gravitational center of the first group and one at the gravitational center of the other). The tree algorithms consist of two phases. In the first phase, an oct-tree is constructed which hierarchically partitions the particles into many smaller clusters. In the second phase, the oct-tree is used to compute an approximation to the force field. In most simulations, the particles are almost homogeneously distributed. In this case, the oct-tree for an n-particle cluster has a height of $\Theta(\log n)$. The oct-tree was built using the following top–down approach. The root node corresponds to a computation box (a cube) big enough to contain all the particles in the given cluster. The n particles are inserted to the root of the tree one by one. Whenever a node in the tree has two or more particles, the corresponding computation box is subdivided into 8 smaller computation boxes, which correspond to the 8 children of the current node. The particles in the current node are then inserted to the children nodes according to there spatial positions. It is clear that $O(n \log n)$ time is required to build the oct-tree this way if the height of the tree is bounded by $O(\log n)$. Both the Appel's algorithm and the Barnes and Hut algorithm require $O(n \log n)$ time to build the tree for homogeneously distributed clusters.

In Section 2, we show that the oct-tree can be constructed bottom-up in $O(n)$ time. In Section 3, we show that the force field can be computed in $O(n)$ time as well. Throughout this paper, we make the following assumption on the distribution of the particles.

Assumption 1.1. *There exist two positive constants c_1 and c_2 such that the minimum inter-particle distance is at least c_1 and the maximum inter-particle distance is smaller than $c_2 n^{1/3}$.*

Assumption 1.1 is highly believed to be true for most applications and is supported by many computer simulations. For the Lennard–Jones cluster, it is proved that the minimum inter-particle distance has positive lower bound which is independent on the number of particles in the cluster [9].

2. Building the oct-tree bottom-up in O(n) time

The top–down construction of the oct-tree described in the previous section is so simple that most researchers assume the existence of the oct-tree in the study of force field evaluation algorithms. However, we have to be able to build the oct-tree faster if we are looking for an asymptoticly faster algorithm. Such an algorithm is presented in this section through a bottom-up approach.

2.1. Data structures

A *computation box* is defined by a point `base` in three-dimensional space and a positive number `size`. Let `baseX`, `baseY`, `baseZ` be the coordinates of point `base`. Then the computation box defined by `base` and `size` is

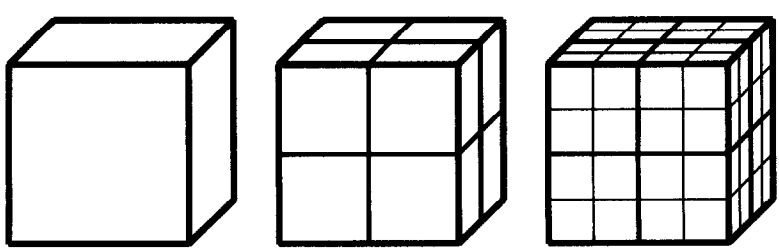
$$[baseX,\ baseX + size) \times [baseY,\ baseY + size) \times [baseZ,\ baseZ + size). \qquad (2.1)$$

Note that each interval in the Cartesian product (2.1) is closed on the left but open on the right. This convention will make the partition of a computation box much easier as we will see later. A computation box is illustrated in Fig. 1(a). When a computation box is partitioned, we obtain 8 non-intersecting computation boxes of equal size whose union is the original computation box. An example is illustrated in Fig. 1(b).

We will make reference to the following data structure during our description of the algorithm:

```
typedef struct _node{
  struct _node  *parent;   struct _node  *child[8];
  int     isLeaf;  int     weight;  int     pindex;
  double  coordX;  double  coordY;  double  coordZ;
  double  forceX;  double  forceY;  double  forceZ;
  double   baseX;  double   baseY;  double   baseZ;  double  size;
}NODE;
```

Each node in the oct-tree is of type `NODE`. For every node in the tree, `weight` is the number of particles contained in the corresponding computation box. If `weight` is 0 or



(a) level 0 (b) level 1 (c) level 2

Fig. 1. Computation boxes associated with the first 3 levels of the oct-tree.

1, we have a leaf node and the field `isLeaf` is 1. If `weight` is 2 or larger, we have an interior node and the field `isLeaf` is 0. For an interior node, the fields `coordX`, `coordY`, `coordX` represent the coordinates of the gravitational center of the particles contained in the computation box corresponding to this node. The fields `baseX`, `baseY`, `baseZ` and `size` define the computation box corresponding to the node in the tree. The fields `forceX`, `forceY` and `forceZ` are used in the evaluation of force field whose use will be discussed later. For a leaf node whose `weight` is 1, `pindex` is the index of the unique particle that is contained in the computation box corresponding to the leaf node. The field `parent` contains a pointer to the parent node in the tree. For the root node, `parent` is `NULL`. For any interior node, `child[j]` is a pointer to the jth child of .the current node ($j = 0, 1, \ldots, 7$). We make the assumption that every interior node has exactly 8 children whose computation boxes have the same size, which is half of the size of the computation box of the current node. If the computation box of the current node is given in (2.1), then the computation boxes for `child[0]`, `child[1]`, ..., `child[7]` are defined as follows:

$$[baseX, baseX + \tfrac{1}{2}size) \times [baseY, baseY + \tfrac{1}{2}size) \times [baseZ, baseZ + \tfrac{1}{2}size),$$

$$[baseX, baseX + \tfrac{1}{2}size) \times [baseY, baseY + \tfrac{1}{2}size) \times [baseZ + \tfrac{1}{2}size, baseZ + size),$$

$$[baseX, baseX + \tfrac{1}{2}size) \times [baseY + \tfrac{1}{2}size, baseY + size) \times [baseZ, baseZ + \tfrac{1}{2}size),$$

$$[baseX, baseX + \tfrac{1}{2}size) \times [baseY + \tfrac{1}{2}size, baseY + size) \times [baseZ + \tfrac{1}{2}size, baseZ + size),$$

$$[baseX + \tfrac{1}{2}size, baseX + size) \times [baseY, baseY + \tfrac{1}{2}size) \times [baseZ, baseZ + \tfrac{1}{2}size),$$

$$[baseX + \tfrac{1}{2}size, baseX + size) \times [baseY, baseY + \tfrac{1}{2}size) \times [baseZ + \tfrac{1}{2}size, baseZ + size),$$

$$[baseX + \tfrac{1}{2}size, baseX + size) \times [baseY + \tfrac{1}{2}size, baseY + size) \times [baseZ, baseZ + \tfrac{1}{2}size),$$

$$[baseX + \tfrac{1}{2}size, baseX + size) \times [baseY + \tfrac{1}{2}size, baseY + size) \times [baseZ + \tfrac{1}{2}size, baseZ + size).$$

Note that the partition of a computation box takes constant time. The computation box in Fig. 1(a) is partitioned to 8 smaller computation boxes in Fig. 1(b), which in turn are partitioned to a total of 64 even smaller computation boxes in Fig. 1(c).

2.2. *Building the tree bottom-up*

We assume that the n particles are given in an array of points so that `part[i].x`, `part[i].y`, `part[i].z` represent the coordinates of particle i ($i = 0, 1, 2, \ldots, n-1$).

In $O(n)$ time, we can compute the base point of the computation box for the root node by computing the minimum of the coordinates of the n particles for each of the

three dimensions. Similarly, we can compute the maximum of the coordinates of the n particles for each of the three dimensions in $O(n)$ time. By Assumption 1.1, we can now decide the size of the smallest computation box as well as the size of the largest computation box, in constant time. By then, we should know an $O(\log n)$ upper bound on the height of the oct-tree. Therefore, we can dynamically allocate space for every possible tree node. We assume that all the fields of a tree node are initialized to zero at the time the memory is allocated. The total space allocated is $O(n)$ as will be analyzed after the presentation of the algorithm.

Instead of inserting the particles to the tree from the root node, we insert the particles directly to the nodes corresponding to the smallest computation boxes. We then pass information from one layer of the tree to the layer above, starting from the bottom layer. Although there are $\Theta(\log n)$ layers of the tree, the amount of time required is decreased by a factor of 8 every time we move up one layer. This is the key to achieving the $O(n)$ time complexity. The complete algorithm is presented in Fig. 2.

2.3. *Analysis of the oct-tree construction algorithm*

Note that each node at level-*maxlevel* in the oct-tree constructed by Algorithm 2.1 has a computation box whose size is $\delta = \sqrt{3}/3c_1$. Therefore, the largest Euclidean distance between any two points in such a computation box is smaller than c_1. It follows from Assumption 1.1 that no two particles can fall into a computation box of at level-*maxlevel*. Again by Assumption 1.1, all n particles are contained in the computation box of the root node.

Since there are 8^l tree nodes at level-l of the tree for $(l = 0, 1, \ldots, maxlevel)$, the number of tree nodes required is

$$\sum_{l=0}^{maxlevel} 8^l = \frac{8^{maxlevel+1} - 1}{8 - 1} \leqslant \frac{8^{maxlevel+1}}{7} \leqslant \frac{192\sqrt{3}}{7} \left(\frac{c_2}{c_1}\right)^3 n, \tag{2.2}$$

where the last inequality follows from Assumption 1.1 and the definition of δ and *maxlevel*. Therefore, our algorithm requires $O(n)$ memory.

In the following, we will analyze the time complexity of Algorithm 2.1. Since there are n particles, the computation of *Xmin*, *Xmax*, *Ymin*, *Ymax*, *Zmin*, *Zmax* requires $\Theta(n)$ time. After the above quantities are computed, *maxlavel* can be computed in constant time by taking a base 2 logarithm. Therefore, Step_1 requires $\Theta(n)$ time.

In Step_2, $\Theta(n)$ space is dynamically allocated. We assume that every field of each tree node is initialized to zero. This may take up to $O(n)$ time. In Step_3, constant amount of time is spent on each particle. Therefore, $\Theta(n)$ time is required in this step. Note that the computation of the gravitational center of the particles contained in the computation box can be done in constant time once we know the field `weight` and the gravitational center for each of its 8 children. Therefore, constant amount of time is spent on each tree node in Step_4. Therefore, $\Theta(n)$ time is required in this step. To summarize, we have proved the following theorem.

Algorithm 2.1 (part 1) {Building the oct-tree bottom-up.}

Step_1 {Compute the root box}

Let $Xmin := \min_{i=1,n} part[i-1].x$; $Ymin := \min_{i=1,n} part[i-1].y$; $Zmin := \min_{i=1,n} part[i-1].z$. Let $Xmax := \max_{i=1,n} part[i-1].x$; $Ymax := \max_{i=1,n} part[i-1].y$; $Zmax := \max_{i=1,n} part[i-1].z$. Let $\delta := \frac{\sqrt{3}}{3}c_1$. Let *maxlevel* be the smallest positive integer such that $\delta 2^{maxlevel} > \max\{Xmax - Xmin, Ymax - Ymin, Zmax - Zmin\}$. Let $\Delta := \delta \times 2^{maxlevel}$.

Step_2 {Allocate space}

We will use a three dimensional array of NODE for the nodes on each level of the oct-tree. Let $tree[l]$ be a pointer to the three dimensional array of NODE with $2^l \times 2^l \times 2^l$ elements ($l = 0, 1, \ldots, maxlevel$). It is clear that we require to allocate $\Theta(n)$ space because there are 8^l tree nodes on level-l of the tree. These arrays are dynamically allocated at this time.

Step_3 {Construct the leaf nodes}

for $p = 1$ **to** n **do**

Let $i = \lfloor \frac{part[p-1].x - Xmin}{\delta} \rfloor$; $j = \lfloor \frac{part[p-1].y - Ymin}{\delta} \rfloor$; $k = \lfloor \frac{part[p-1].z - Zmin}{\delta} \rfloor$.

$tree[maxlevel] \rightarrow node[i][j][k].size := \delta$;

$tree[maxlevel] \rightarrow node[i][j][k].baseX := Xmin + i\delta$;

$tree[maxlevel] \rightarrow node[i][j][k].baseY := Ymin + j\delta$;

$tree[maxlevel] \rightarrow node[i][j][k].baseZ := Zmin + k\delta$;

$tree[maxlevel] \rightarrow node[i][j][k].weight := 1$;

$tree[maxlevel] \rightarrow node[i][j][k].pindex := p$;

$tree[maxlevel] \rightarrow node[i][j][k].coordX := part[p-1].x$;

$tree[maxlevel] \rightarrow node[i][j][k].coordY := part[p-1].y$;

$tree[maxlevel] \rightarrow node[i][j][k].coordZ := part[p-1].z$.

endfor

Fig. 2. Building the oct-tree bottom-up (part 1).

Theorem 2.1. *Algorithm* 2.1 *builds the oct-tree for n particles using* $\Theta(n)$ *time and* $\Theta(n)$ *space, provided that the particles satisfies Assumption* 1.1. *The constant behind the asymptotic notation is proportional to* $(c_2/c_1)^3$.

Note that although we have allocated space for a full oct-tree, the actual oct-tree may not be a full oct-tree in most cases. As a result, the leaf nodes of the oct-tree may be at different levels of the tree.

Note also that under the same assumption on the distribution of the particles, the top-down construction of the oct-tree requires $O(n \log n)$ time [2, 3]. In our bottom-up construction, we are allocating space for some tree nodes which will never be used (i.e., the descendants of a node whose weight is 1). We should note that the asymptotic

Algorithm 2.1 (part 2) {Building the oct-tree bottom-up.}

Step_4 {Building the tree bottom-up}

for $l := maxlevel - 1$ **downto** 0 **do**

for $i := 0$ **to** $2^l - 1$ **do for** $j := 0$ **to** $2^l - 1$ **do for** $k := 0$ **to** $2^l - 1$ **do**

$tree[l] \rightarrow node[i][j][k].child[0] := tree[l+1] \rightarrow node[2i+0][2j+0][2k+0];$

$tree[l] \rightarrow node[i][j][k].child[1] := tree[l+1] \rightarrow node[2i+0][2j+0][2k+1];$

$tree[l] \rightarrow node[i][j][k].child[2] := tree[l+1] \rightarrow node[2i+0][2j+1][2k+0];$

$tree[l] \rightarrow node[i][j][k].child[3] := tree[l+1] \rightarrow node[2i+0][2j+1][2k+1];$

$tree[l] \rightarrow node[i][j][k].child[4] := tree[l+1] \rightarrow node[2i+1][2j+0][2k+0];$

$tree[l] \rightarrow node[i][j][k].child[5] := tree[l+1] \rightarrow node[2i+1][2j+0][2k+1];$

$tree[l] \rightarrow node[i][j][k].child[6] := tree[l+1] \rightarrow node[2i+1][2j+1][2k+0];$

$tree[l] \rightarrow node[i][j][k].child[7] := tree[l+1] \rightarrow node[2i+1][2j+1][2k+1];$

Also set the *parent* field for each of the 8 children nodes;

$tree[l] \rightarrow node[i][j][k].baseX := Xmin + i2^{maxlevel-l}\delta;$

$tree[l] \rightarrow node[i][j][k].size := 2^{maxlevel-l}\delta;$

$tree[l] \rightarrow node[i][j][k].baseY := Ymin + j2^{maxlevel-l}\delta;$

$tree[l] \rightarrow node[i][j][k].baseZ := Zmin + k2^{maxlevel-l}\delta;$

Let $tree[l] \rightarrow node[i][j][k].weight$ be the sum of the weights of its children;

Let $tree[l] \rightarrow node[i][j][k].coordX$, $tree[l] \rightarrow node[i][j][k].coordY$, and $tree[l] \rightarrow node[i][j][k].coordZ$ be the coordinates of the weighted center of the particles contained in the computation box of the current node;

if $tree[l] \rightarrow node[i][j][k].weight == 0$ **then**

$tree[l] \rightarrow node[i][j][k].isLeaf := 1;$

elseif $tree[l] \rightarrow node[i][j][k].weight == 1$ **then**

$tree[l] \rightarrow node[i][j][k].isLeaf := 1;$

Let $tree[l] \rightarrow node[i][j][k].pindex$ be the index of the only particle contained in the current computation box;

endif

endfor endfor endfor

endfor

Fig. 2. Building the oct-tree bottom-up (part 2).

memory requirements for the top-down algorithm and the bottom-up algorithm are both $O(n)$. However, the time complexity of the bottom-up algorithm is a $\Theta(\log n)$ factor lower than that of the top–down algorithm. In the next section, we will show that Appel's algorithm for computing force field of a cluster of n particles can be implemented in $O(n)$ time after the oct-tree is constructed. Therefore, the improved time complexity of the construction of the oct-tree has great impact on the simulation of large clusters.

3. Computing force fields top-down in $O(n)$ time

Given a cluster of n particles, we need to compute the potential energy function and the force exerted on each particle by the other particles. In many applications, the potential energy function of a cluster is the sum of the pair-wise potential functions.

Let p_1 and p_2 be the positions of two particles of unit charge each, the Lennard–Jones potential function between this pair of particles is defined by

$$f_{LJ}(p_1, p_2) = \frac{\sigma_1}{\|p_2 - p_1\|^{12}} - \frac{\sigma_2}{\|p_2 - p_1\|^6}, \tag{3.1}$$

where $\|\bullet\|$ stands for the Euclidean norm and σ_1 and σ_2 are given positive constants. This potential energy function is widely used in molecular conformation, molecular dynamics and protein folding. In other applications, the following gravitational potential

$$f_G(p_1, p_2) = \frac{\sigma_3}{\|p_2 - p_1\|} \tag{3.2}$$

is often used, where σ_3 is a given positive constant. In both cases, the potential energy function approaches zero as the distance between the two particles approaches to infinity. In both cases, the "force" exerted on p_1 by p_2 is computed as the negative of the gradient of the potential energy function with respect to p_1. These are given by

$$\left(\frac{-12\sigma_1}{\|p_2 - p_1\|^{14}} - \frac{-6\sigma_2}{\|p_2 - p_1\|^8}\right)(p_2 - p_1) \tag{3.3}$$

and

$$\frac{-\sigma_3}{\|p_2 - p_1\|^3}(p_2 - p_1) \tag{3.4}$$

for the Lennard–Jones pair potential and the gravitational pair potential, respectively.

Since there are $n(n-1)/2$ pairs for a cluster of n particles, conventional algorithm for computing the potential energy function and force field requires $O(n^2)$ time.

In 1985, Appel [2] proposed a divide and conquer algorithm for computing the force field of a gravitational cluster. His algorithm has a proved time complexity of $O(n \log n)$. Appel's algorithm is based on the following idea: Given two clusters of particles consisting of n_1 and n_2 unit weight particles each, there are $n_1 \times n_2$ particles

pairs with one particle from the first cluster and the other particle from the other cluster. If the two clusters are *well separated* (i.e., the ratio of the maximum diameter of the clusters over the distance between the clusters is small), we may consider the first cluster as a *big particle* located at the gravitational center P_1 of the first cluster with a mass of n_1 and consider the second cluster as another *big particle* located at the gravitational center P_2 of the second cluster with a mass of n_2. We may then approximate the total interactions between particles from the first cluster and particles from the second cluster by the following weighted pair potential:

$$F_G(P_1, n_1; P_2, n_2) = n_1 n_2 \frac{\sigma_3}{\|P_2 - P_1\|}. \tag{3.5}$$

The negative of the gradient of the above function with respect to P_1 is

$$n_1 n_2 \frac{-\sigma_3}{\|P_2 - P_1\|^3}(P_2 - P_1). \tag{3.6}$$

Therefore, the force exerted on each particle in the first cluster by all the particles in the second cluster can be approximated by

$$n_2 \frac{-\sigma_3}{\|P_2 - P_1\|^3}(P_2 - P_1) \tag{3.7}$$

since there are n_1 particles in the first cluster. Similarly, the force exerted on each particle in the second cluster by all the particles in the first cluster can be approximated by

$$n_1 \frac{-\sigma_3}{\|P_2 - P_1\|^3}(P_1 - P_2). \tag{3.8}$$

In the case where Lennard–Jones potential energy function is used, the force exerted on each particle in the first cluster by all the particles in the second cluster can be approximated by

$$n_2 \left(\frac{-12\sigma_1}{\|P_2 - P_1\|^{14}} - \frac{-6\sigma_2}{\|P_2 - P_1\|^8} \right)(P_2 - P_1) \tag{3.9}$$

and the force exerted on each particle in the second cluster by all the particles in the first cluster can be approximated by

$$n_1 \left(\frac{-12\sigma_1}{\|P_2 - P_1\|^{14}} - \frac{-6\sigma_2}{\|P_2 - P_1\|^8} \right)(P_1 - P_2). \tag{3.10}$$

In this way, we can spend constant time to compute an approximation to the potential that requires $\Theta(n_1 \times n_2)$ time in the conventional method. If the ratio of the maximum of the radii of the clusters over the distance between the clusters is δ, the relative error in this approximation is $O(\delta^2)$ [2].

The performance of Appel's algorithm depends on the parameter which defines well separateness. If this parameter is close to 0, we have more accuracy but need more computing time. If this parameter is close to 1, we have less accuracy but need less

computing time. A generic description of Appel's algorithm is given in the next section. We will prove that Appel's algorithm actually requires only $O(n)$ time when the oct-tree is given.

3.1. The algorithm

After the oct-tree is constructed, Appel's algorithm can be implemented using the procedures `oneNODE`, `twoNODES` and `pushDOWN`. We assume that there is a global variable `FUNC` which is initialized to 0 and is used to accumulate the potential energy function of the cluster. We also assume that the fields `forceX`, `forceY` and `forceZ` at every tree node are all initialized to 0 before the computation. These fields are used to hold partial values of the force field during the computation.

The function `distance(A, B)` measures the distance between the centers of the two computation boxes associated with the two tree nodes pointed to by `A` and `B`. It is convenient to use the L_∞ norm, i.e., the largest difference in the three dimensions. The procedure `compGRAD(A, B)` treats the particles contained in the computation box of `A` as a big particle at the gravitational center of these particles with weight equal to `A->weight` and treats the particles contained in the computation box of `B` as a big particle at the gravitational center of these particles with weight equal to `B->weight`. It then approximates the `A->weight * B->weight` pair potential energy by a single weighted pair potential (3.5). It also computes the negative of the gradient of this weighted pair potential with respect to the gravitational center of `A` and stores this information in the fields `A->forceX`, `A->forceY` and `A->forceZ`. The opposite of this force is stored in the fields `B->forceX`, `B->forceY` and `B->forceZ`.

The computation of force field is initiated by the procedure call `oneNODE(root, delta)` where `root` is a pointer to the root of the oct-tree and `delta` is the well-separateness parameter. This procedure call is followed by the procedure call `pushDOWN(root)`. When the second call returns, the total potential energy function can be found in the global variable `FUNC` and the force field exerted on each particle can be found in the fields `forceX`, `forceY` and `forceZ` at the corresponding leaf node of the oct-tree. The memory can now be freed using the array of pointers `level[]` so that there is no memory leek during the computation.

3.2. Time complexity

To analyze the time complexity of the force evaluation algorithm, all we need to do is to estimate the number of procedure calls of `oneNODE`, `twoNODE` and `pushDOWN`.

Note that every call to `pushDOWN` is associated with a unique tree node whose `weight` field is 2 or larger. Therefore, the number of calls to `pushDOWN` is bounded by the number of nodes in the tree, which in turn is bounded by $(192\sqrt{3}/7)(c_2/c_1)^3 n$. Similarly, the number of calls to `oneNODE` is also bounded by $192\sqrt{3}/7(c_2/c_1)^3 n$.

Now, let us estimate the number of calls to `twoNODES`. Note that the first two parameters to any `twoNODES` are always pointers to tree nodes at the same level. For any pair of tree nodes `A` and `B` at the same level of the oct-tree, the ratio of the radius

```
void oneNODE(NODE *A, double delta)
{ int i, j;

  if (A->isLeaf == 0){
    for (i=0; i<8; i++) oneNODE(A->child[i], delta);
    for (i=0; i<7; i++)
      for (\kern1ptj=i+1; j<8; j++)
        twoNODES(A->child[i], A->child[j], delta);
  }
}

void twoNODES(NODE *A, NODE *B, double delta)
{ double d; int i, j;

  if (A->weight == 0 \Vert B->weight == 0) return;
  d = distance(A, B);
  if (((A->size/d) <= delta) && ((B->size/d) <= delta))
    compGRAD(A, B);
  else
    for (i=0; i<8; i++)
      for (\kern1ptj=0; j<8; j++)
        twoNODES(A->child[i], B->child[j], delta);
}

void pushDOWN(NODE *A)
{ int i;
  while (A->weight >= 2){
    for (i=0; i<8; i++){
      A->child[i].forceX += A->forceX / A->weight;
      A->child[i].forceY += A->forceY / A->weight;
      A->child[i].forceZ += A->forceZ / A->weight;
      pushDOWN(A->child[i]);
    }
  }
}
```

Fig. 3. Computing the potential energy function and force field top–down.

of the computation box of A over the distance between the centers of the computation boxes of A and B is $1/(2d+2)$, where d is the number of computation boxes of the same size between the computation boxes for A and B. Therefore, if

$$\delta = \frac{1}{2d+2} \quad \text{or} \quad d = \frac{1}{2\delta} - 1, \tag{3.11}$$

the number of twoNODES calls involving two nodes at level-l of the oct tree is bounded by

$$8(2d+1)^3 8^l, \quad l=0,1,2,\ldots,maxlevel. \tag{3.12}$$

Therefore, the total number of calls to twoNODES is bounded by $O((c_2/c_1 \times 1/\delta)^3 n)$. Therefore, we have proved the following bound on the time complexity for force field calculation.

Theorem 3.1. *Given a cluster of n particles satisfying Assumption* 1.1, *the force field can be computed using Appel's algorithm in* $O((c_2/c_1 \times 1/\delta)^3 n)$ *time, where* c_1 *and* c_2 *are the distribution parameters and* δ *is the well-separateness parameter.*

The about estimate of calls to twoNODES was given by Esselink [5]. Although Esselink used the assumption that there is a particle in the computation box for every node at level-*maxlevel*, the argument directly translates to the above analysis. In [4], Callahan and Kosaraju also proved that a size $O(n)$ sequence of *well-separated decomposition* can be computed in $O(n)$ time once a *fair-split* tree is constructed. A fair-split tree for n particles can be constructed in $O(n \log n)$ time using the algorithm of [4], without any restriction on the distribution of the particles. However, Algorithm 2.1 is the first linear time algorithm for oct-tree construction.

Remark 3.1. In the Barnes and Hut algorithm, one approximates the interactions between a single particle and a group of particles using a single computation. Since every particle interacts with $O(\log n)$ well-separated groups of particles, the computation time required by the Barnes and Hut algorithm is still $O(n \log n)$. This is a major difference between Appel's algorithm and the Barnes and Hut algorithm.

Remark 3.2. Assumption 1.1 is essential to the $O(n)$ time oct-tree construction algorithm. Without any assumption on the distribution of the particles, $\Omega(n \log n)$ is lower bound on the construction of the oct-tree for n particles. Consider the case where all n particles lie on the X-axis. Suppose that one can construct the oct-tree of n particles in $T(n)$ time, then the number of tree nodes is $O(T(n))$. Taking an in-order traversal of the tree sorts the n particles. Since the in-order traversal takes $\Theta(T(n))$ time, this shows that $\Theta(T(n)) = \Omega(n)$, under the algebraic comparison tree model. Which shows that $T(n) = \Omega(n)$.

4. Conclusions

In this paper, we have presented an $O(n)$ time variant of Appel's algorithm for force field evaluation in N-body simulations. A key to this improved complexity is an $O(n)$ time bottom-up construction of the oct-tree which was constructed top–down using $O(n \log n)$ time in previous studies. We have also studied the dependency of the

constant behind the asymptotic notation on the distribution parameters c_1 and c_2 and on the well-separateness parameter δ. This analysis is important because good software for these evaluations is badly needed in practice. Computational studies of the proposed algorithm will be reported in a forthcoming paper.

This research was supported in part by a Research Initiation Award of the National Science Foundation under Grant ASC-9409285.

References

[1] R.J. Anderson, Tree data structures for N-body simulation, 37th Annual Symposium of Foundations of Computer Science IEEE (1996) 224–233.
[2] A.W. Appel, An efficient program for many-body simulation, SIAM J. on Sci. Statist. Comput. 6 (1985) 85–103.
[3] J. Barnes, P. Hut, A hierarchical $O(n \log n)$ force-calculation algorithm, Nature 324 (1986) 446–449.
[4] P.B. Callahan, S.R. Kosaraju, A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields, J. ACM 42 (1995) 67–90.
[5] K. Esselink, The order of Appel's algorithm, Inform. Process. Lett. 41 (1992) 141–147.
[6] L. Greengard, The rapid evaluation of potential fields in particle systems, The MIT Press, Cambridge, MA, 1988.
[7] L. Greengard, Fast algorithms for classical physics, Sci. 265 (1994) 909–914.
[8] L. Greengard, V. Rokhlin, A fast algorithm for particle simulations, J. Comput. Phy. 73 (1987) 325–348.
[9] G.L. Xue, Minimum inter-particle distance at global minimizers of Lennard–Jones clusters, J. Global Optim. 11 (1997) 83–90.